CSCI 361 Lecture 9: Context-Free Languages II

Shikha Singh

Announcements & Logistics

- **HW 3** due Wed (Oct 9)
  - Please ensure that any DFA/ Parse tree images attached are clear
  - You can use figure flags to ensure LaTeX places them in the right spot
- Hand in **reading questions # 6** and pick up **reading questions #7**
- **Reminder:** What I did Last Summer Colloquium tomorrow
- CSCI 361 Midterm on **Oct 22 (Tuesday)**:
  - In class exam 75 mins exam
  - Can bring your notes but no screens allowed
  - A textbook will be available for reference
  - Will provide more details about format before exam

Last Time

- Wrapped up regular languages
- Started context-free grammars

Today and Coming Lectures

- More on context-free languages and push-down automata
    - Less focus on automata than regular languages
    - Still good to know
- Non-context-free pumping lemma

Regular Languages are Context-Free

- Let $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA for the regular language L
- We can construct a CFG G for L as follows
    - Make a variable Q_i for each state $q_i \in Q$
    - For each $q_i, q_j \in Q$ and $a \in \Sigma$ such that $\delta(q_i, a) = q_j$ a rule $Q_i \to a\ Q_j$ add a rule $Q_i \to a\ Q_j$
    - Make Q_0 the start variable
    - Add $Q_i \to \varepsilon$ if $q_i \in F$

Regular Grammars

- A CFG is **regular** if any occurrence of a variable on the RHS of a rule is as the rightmost symbol
- If a CFG is regular, there is a DFA that recognizes the same language
  - $Q = V \cup \{f\}$ (A state for each variable plus an accept state)
  - Rule $A \rightarrow aB$ becomes $\delta(A, a) = B$
  - If there is a $A \rightarrow a$ then $\delta(A, a) = f$

CFG for this Language?

- CFG for $L = \{a^i b^j c^k \mid i = j$ or $j = k\}$
- Union of $L_1 = \{a^i b^i c^j | i, j \geq 0\}$ and $L_2 = \{a^i b^j c^j \mid i, j \geq 0\}$

Closure Properties of CFLs

- CFLs are closed under
  - Union
  - Concatenation
  - Kleene star
- Not closer under complement and intersection!

Closure Properties of CFLs

Given $G_1 = (V_1, \Sigma_1, R_1, S_1)$
$G_2 = (V_2, \Sigma_2, R_2, S_2)$

Union: $L(G_1) \cup L(G_2)$ is generated by
$R_1 \cup R_2 \cup \{S \rightarrow S_1, S \rightarrow S_2\}$

NB: Assume that $V_1 - \Sigma_1, V_2 - \Sigma_2$ are disjoint.

Closure Properties of CFLs

Given $G_1 = (V_1, \Sigma_1, R_1, S_1)$
$G_2 = (V_2, \Sigma_2, R_2, S_2)$

Union: $L(G_1) \cup L(G_2)$ is generated by
$R_1 \cup R_2 \cup \{S \to S_1, S \to S_2\}$

Concatenation: $L(G_1)L(G_2)$ is generated by
$R_1 \cup R_2 \cup \{S \to S_1 S_2\}$

NB: Assume that $V_1 - \Sigma_1, V_2 - \Sigma_2$ are disjoint.

Closure Properties of CFLs

Given $G_1 = (V_1, \Sigma_1, R_1, S_1)$

$G_2 = (V_2, \Sigma_2, R_2, S_2)$

Union: $L(G_1) \cup L(G_2)$ is generated by

$R_1 \cup R_2 \cup \{S \rightarrow S_1, S \rightarrow S_2\}$

Concatenation: $L(G_1)L(G_2)$ is generated by

$R_1 \cup R_2 \cup \{S \rightarrow S_1S_2\}$

Kleene $*$: $L(G_1)^*$ is generated by

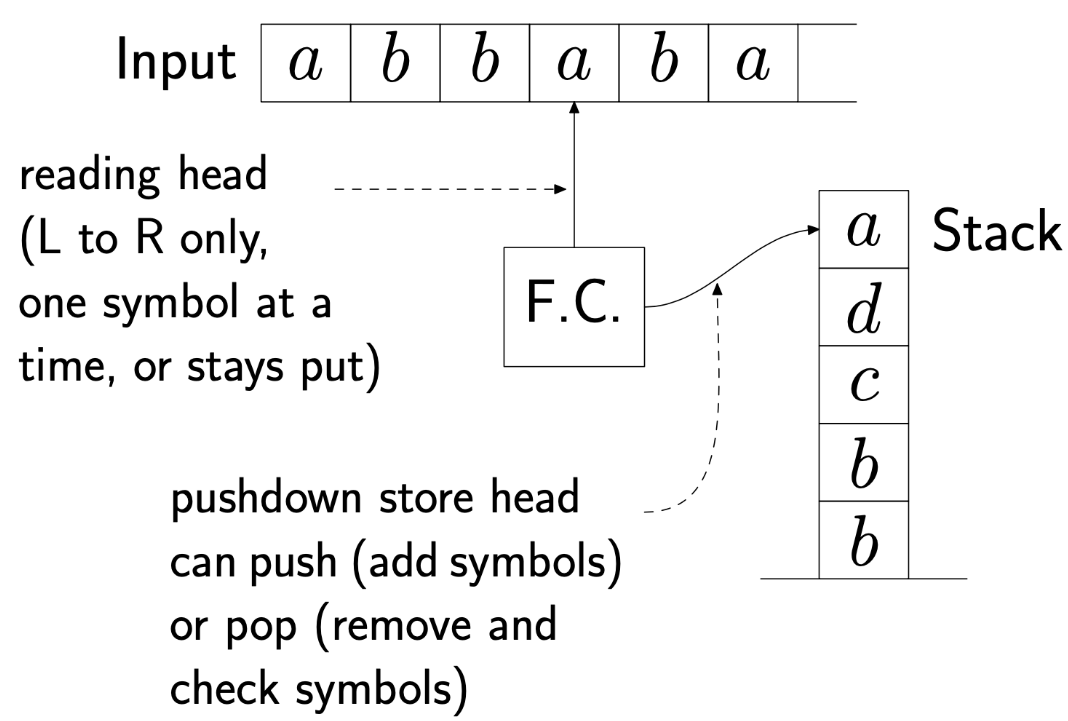
$R_1 \cup \{S \rightarrow e|S \rightarrow S_1S\}$

Automata for CFGs

- Regular Languages : Finite Automata
- Context-free languages: ??

Pushdown Automata

- Basically an NFA with a stack (pushdown store)
- The stack can consist of unlimited number symbols but can only be read and altered at the top:
    - Can only pop symbol from top or push symbol to top



Pushdown Automata Transitions

- Transitions of a PDA have two parts:
  - **State transition** and **stack manipulation** (push/pop)
  - If in state p reading input symbol a and b on the stack, replace b with c on the stack and enter state q
    - $(p, a, b) \to (q, c)$
    - $\delta : Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \to \mathscr{P}(Q \times \Gamma_\varepsilon)$
  - In state diagram arrow goes from $p \to q$ with label $a, b \to c$

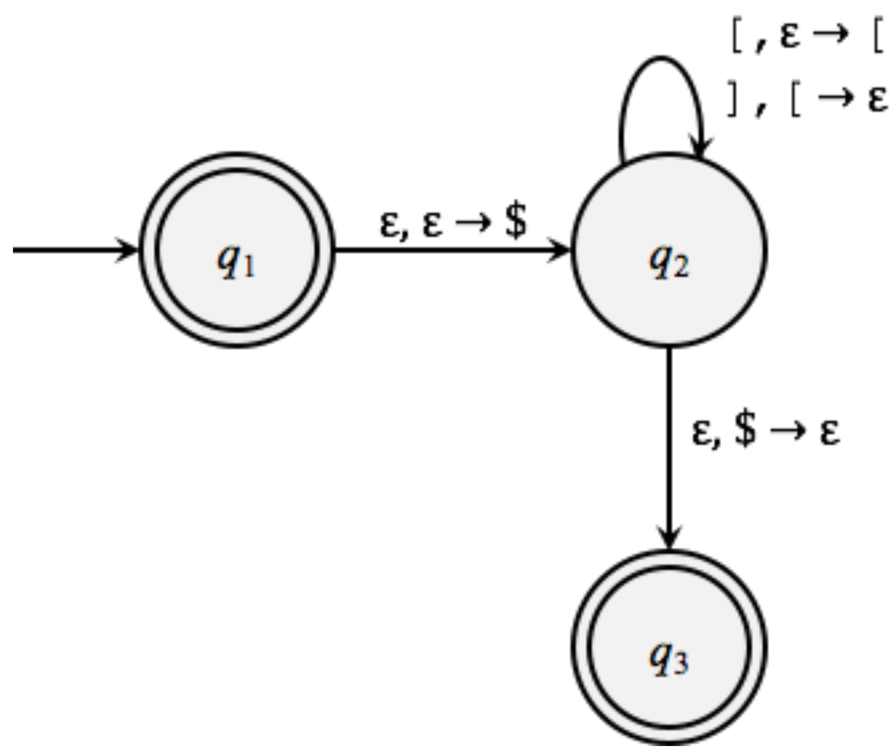
Formal Definition: PDA

- A pushdown automaton is a six tuple $M = (Q, \Sigma, \Gamma, \delta, q_0 F)$ where
  - Q is the finite set of states
  - Σ is a finite alphabet (the input symbols)
  - Γ is a finite tape alphabet (the stack symbols)
  - $\delta : Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \to \mathscr{P}(Q \times \Gamma_\varepsilon)$ is the transition function
  - $q_0 \in Q$ is the initial state and $F \subseteq Q$ is the set of accept states

Example PDA

- Consider the language over $\Sigma = \{[,]\}$ of all strings made up of correctly nested brackets
- CFG for this language: $S \to \varepsilon \mid [S] \mid SS$
- Now lets create a push-down automata for this language
- What to store on the stack?

Example PDA for Balanced Brackets



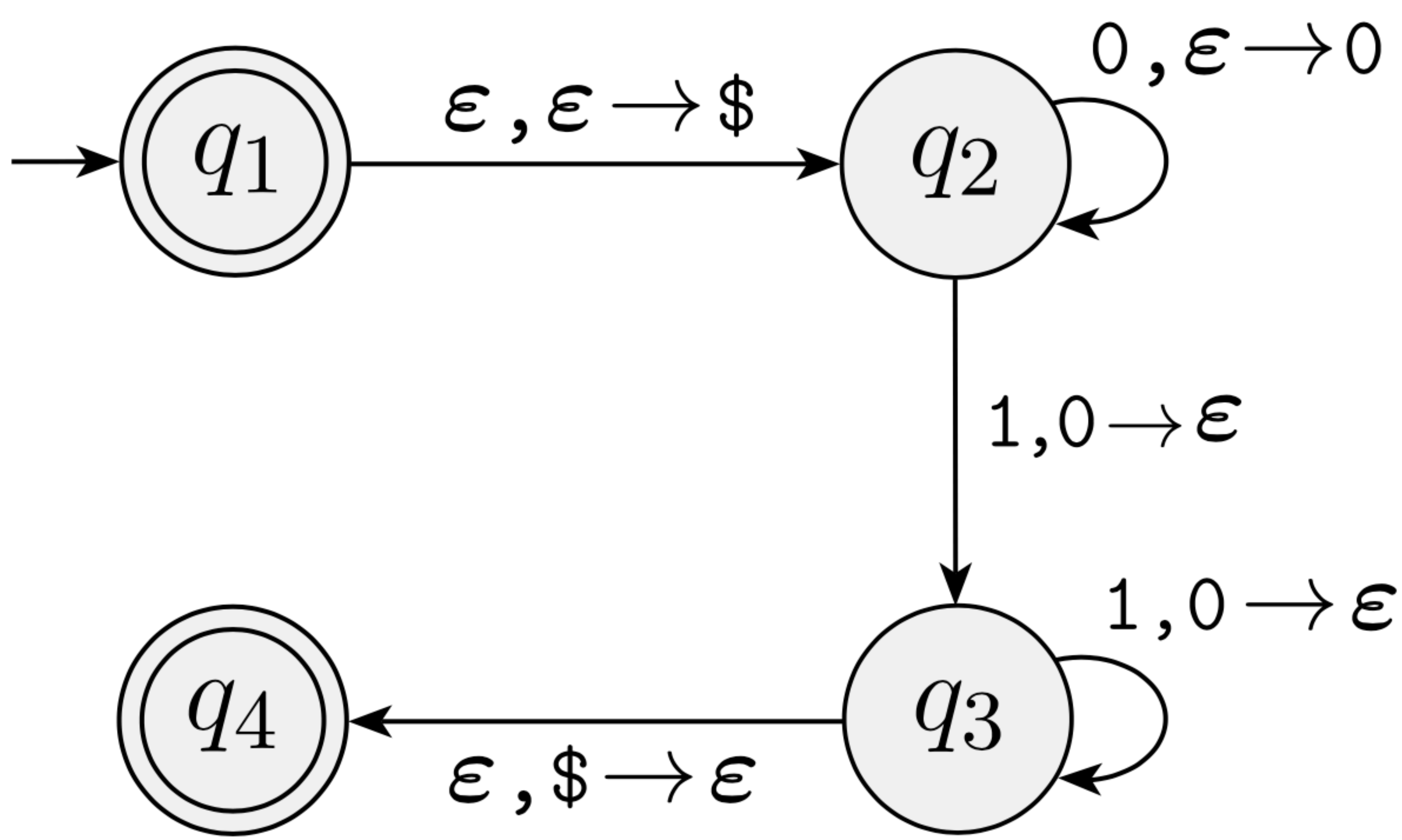
Recall: A transition of the form a, b → z means "if the current input symbol is a and the current stack symbol is b, then follow this transition, pop b, and push the string z"

PDA Acceptance: Informal

- A PDA accepts an input string w if there is a computation that:
  - starts in the start state and empty stack
  - has a sequence of valid transitions
  - at least one computation branch ends in an accept state with an empty stack
- A PDA computation branch "dies off" if
  - no transition matches the input (as in an NFA)
  - no rule matches the stack states
  - any combination of the above
- Language of a PDA: set of all strings that are accepted

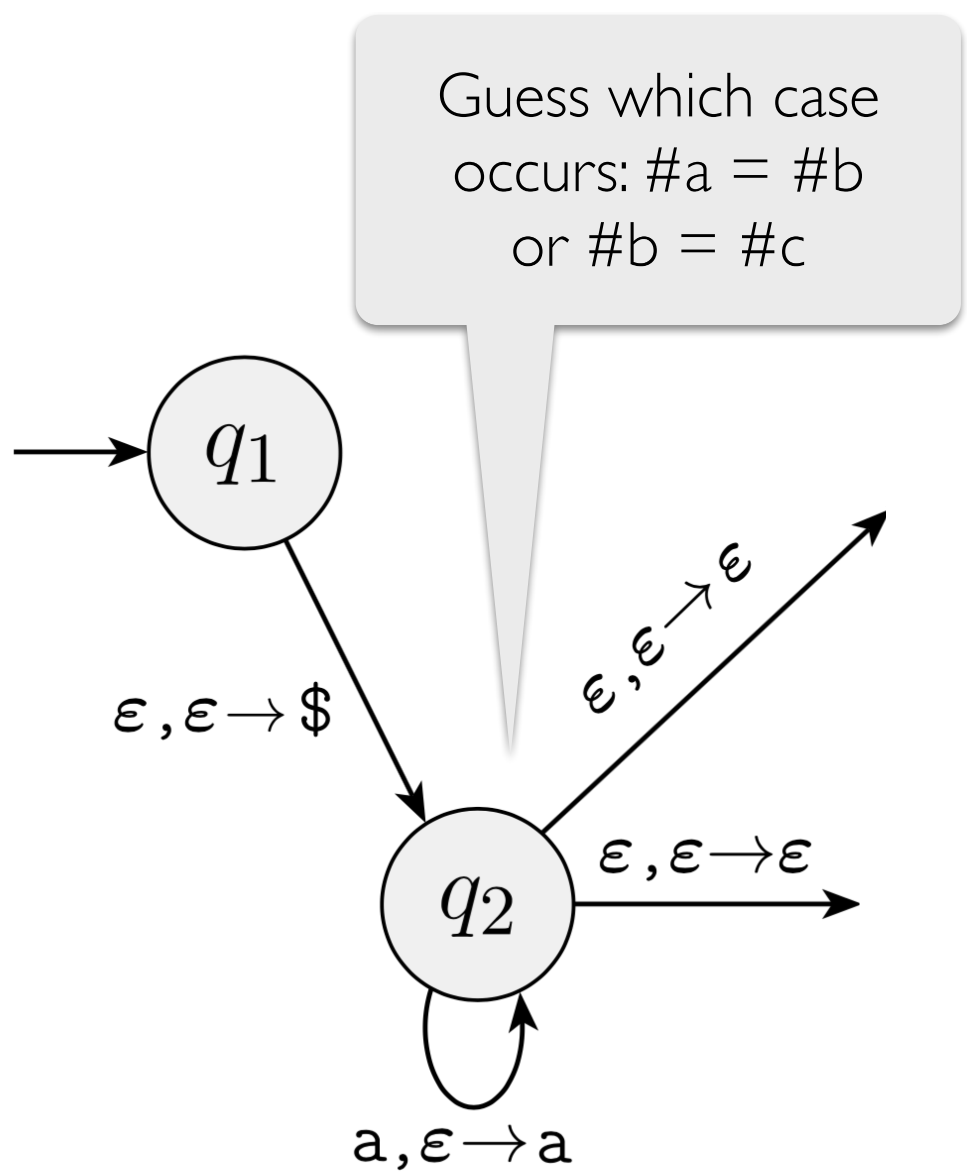
PDA More Examples

- $L = \{0^n 1^n \mid n \geq 0\}$



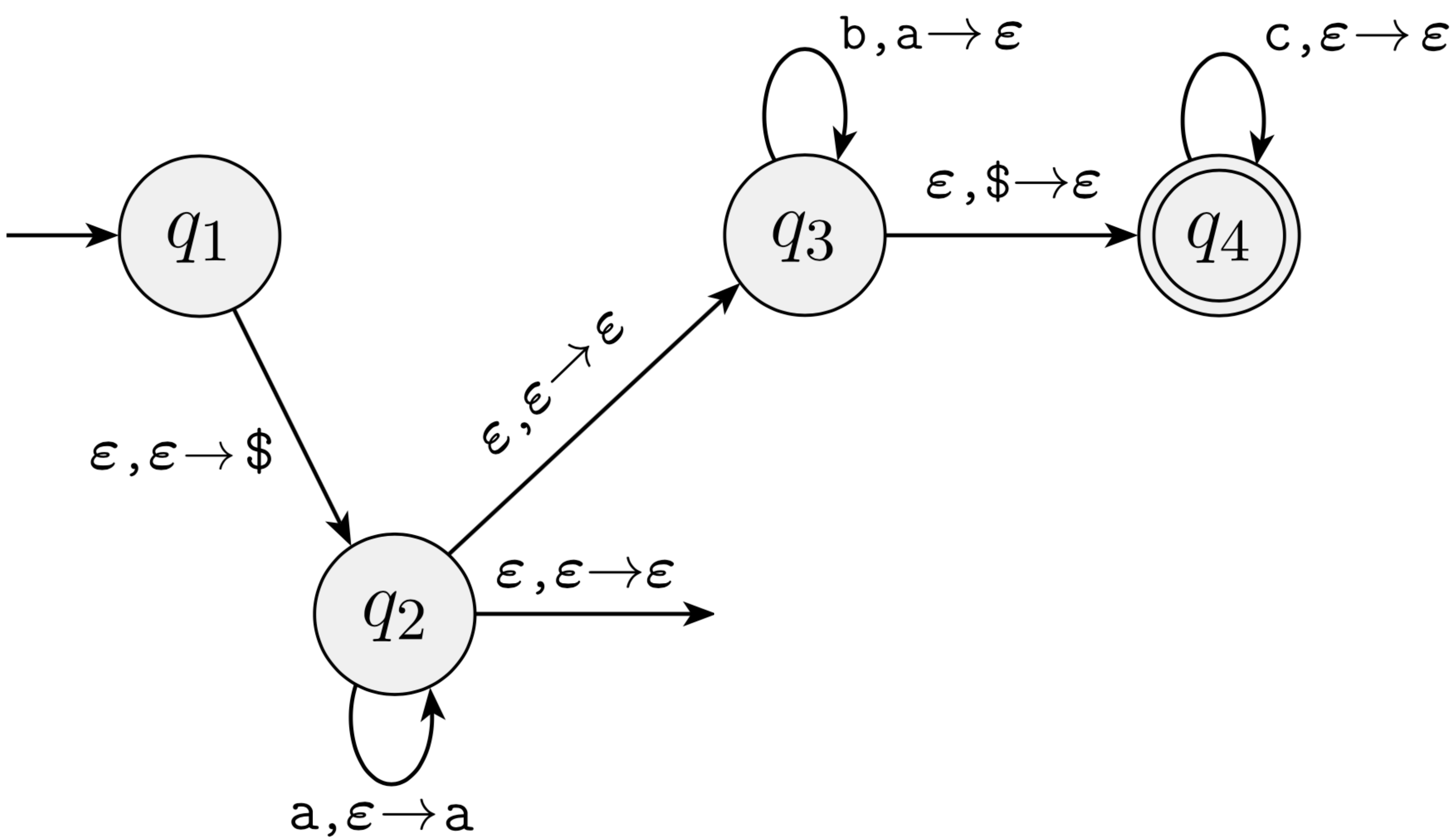
PDA More Examples

- PDA for $L = \{a^i b^j c^k \mid i = j \text{ or } j = k\}$



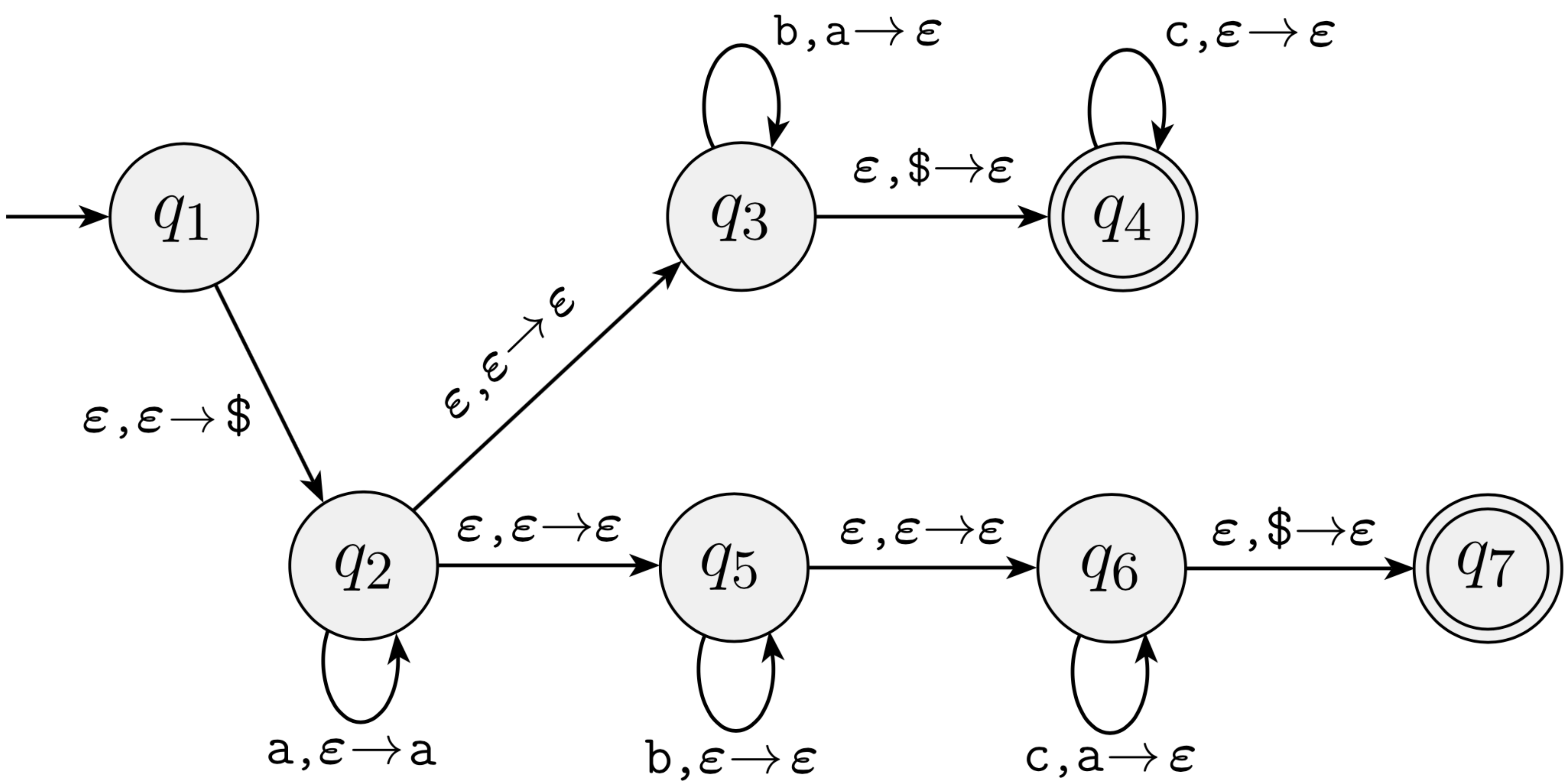
PDA More Examples

- PDA for $L = \{a^i b^j c^k \mid i = j \text{ or } j = k\}$



PDA More Examples

- PDA for $L = \{a^i b^j c^k \mid i = j \text{ or } j = k\}$



Practice Problem

- Draw a PDA for $L = \{ww^R \mid w \in \{0,1\}^*\}$
- Solution is in the book (Sipser 2.1)

Equivalence: CFG $\Longleftrightarrow$ PDA

Theorem. A language is context-free if and only it is recognized by some (non-deterministic) pushdown automaton.

We won't prove this: details are annoying but important to know!

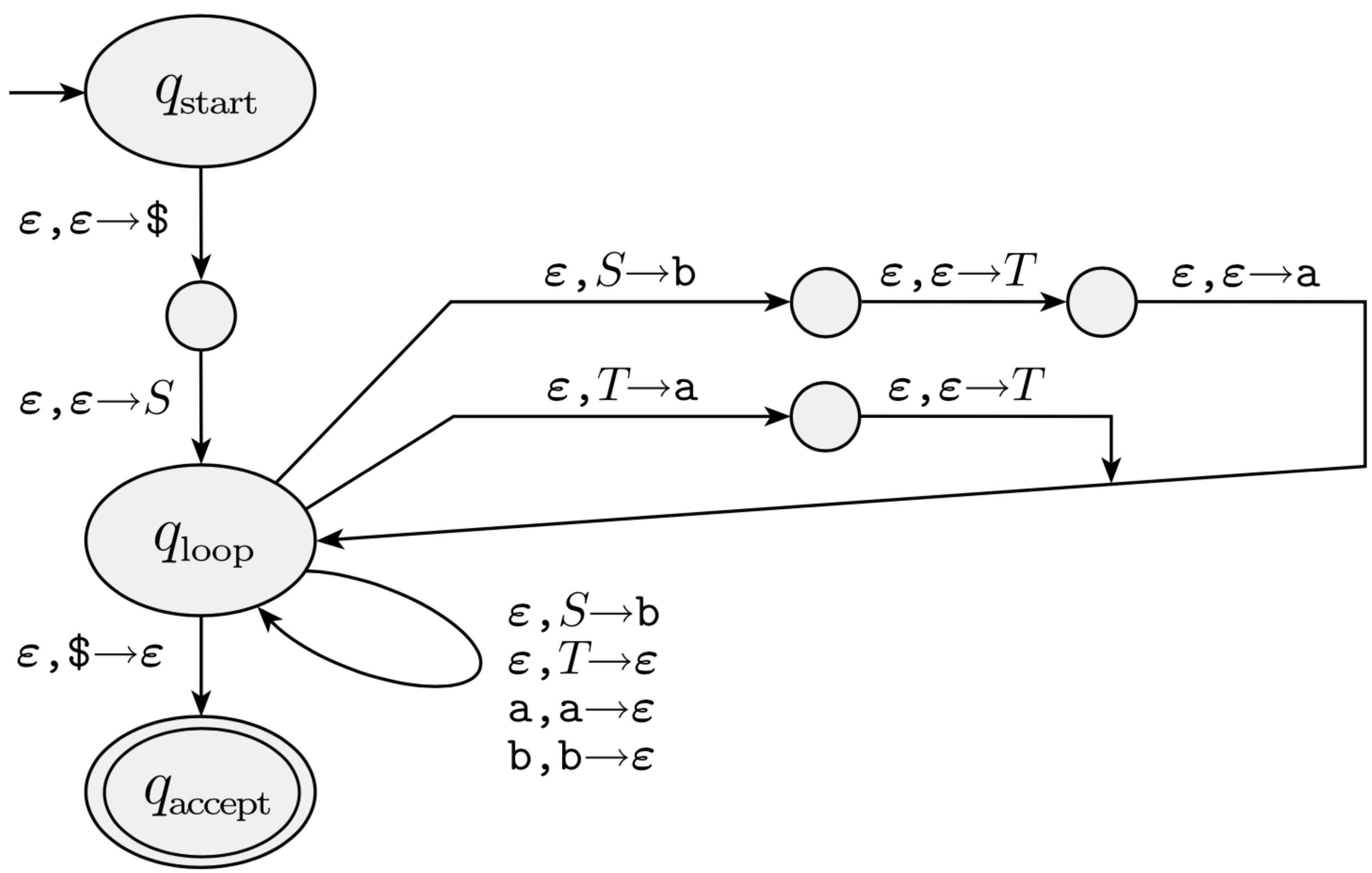
Note: Unlike DFA and NFA, non-deterministic PDAs are more powerful than deterministic PDAs.

Intuition: CFG $\Longrightarrow$ PDA

- Let $G = (V, \Sigma, R, S)$
- Construct a PDA with three main states: **start**, **loop** and **accept** state (some extra states for bookkeeping)
- Start by putting S on the stack
- Each time top of stack is a variable A, guess a rule of the type $A \rightarrow u$ replace A with RHS of the rule
- Each time top of stack is a terminal match it to the current input symbol (computation dies off it they don't match)
- If you reach bottom of stack at any point in a branch, accept

Example: CFG $\Longrightarrow$ PDA

$$S \to \mathtt{a}T\mathtt{b} \mid \mathtt{b}$$
$$T \to T\mathtt{a} \mid \varepsilon$$

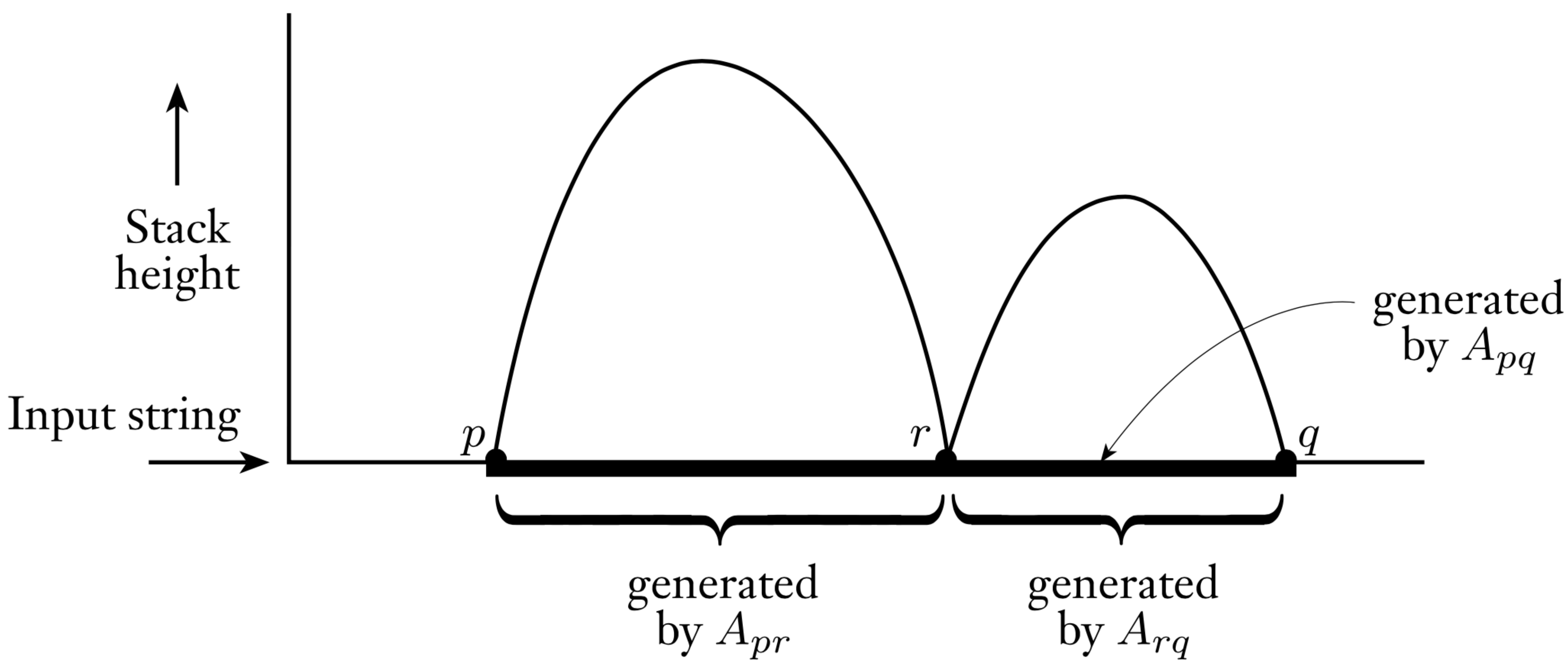


Intuition: PDA $\Longrightarrow$ CFG

- Wlog assume the PDA has one accept state, empties stack before accepting and each move is a push or pop (but not both)
- Let Q be the states of the PDA
- Create variables for each pair of states: $\{A_{pq} \mid p, q \in Q\}$
- A_{pq} generates all strings that take the PDA from p to q starting from an empty stack and ending at an empty stack
  - Such strings can also take PDA from p to q from a non-empty stack returning to exactly the same stack contents
- Start variable is A_{q_0,q_f} where q_0 is start state and q_f is accept state

Intuition: PDA $\Longrightarrow$ CFG

- Add the rules
    - $A_{pq} \to A_{pr}A_{rq}$ for every triple $p, q, r \in Q$
    - $A_{pp} \to \varepsilon$ for $p \in Q$



Intuition: PDA $\Longrightarrow$ CFG

- Finally, if there are rules of the form $(p, a, \epsilon) \to (r, u)$ and $(s, b, u) \to (q, \epsilon)$
- To simulate this add the rule $A_{pq} \to aA_{rs}b$ where PDA goes from p to q after pushing a and s to r after popping b